

# Approximation Algorithms and Heuristics for Classical Planning

Jeremy Frank and Ari Jónsson \*

NASA Ames Research Center †

Mail Stop N269-3; Moffett Field, CA 94035-1000

E-mail: {frank,jonsson}@email.arc.nasa.gov

August 18, 2005

## 1 Introduction

Automated planning has been an active area of research in theoretical computer science and Artificial Intelligence (AI) for over 40 years. Planning is the study of general purpose algorithms that accept as input an *initial state*, a set of desired *goal states*, and a *planning domain model* that describes how *actions* can transform the state. The problem is to find a sequence of actions that transforms the initial state into one of the goal states. Planning is widely applicable, and has been used in such diverse application domains as spacecraft control [MNPW98], planetary rover operations [BJMR05], automated nursing aides [MP02], image processing [GPNV03], computer security [BGHH05] and automated manufacturing [RDF05]. Planning is also the subject of continued and lively ongoing research.

In this chapter, we will present an overview of how approximations and related techniques are used in automated planning. We focus on *classical planning* problems, where states are conjunctions of propositions, all state information is known to the planner, and all action outcomes are deterministic. Classical planning is nonetheless a large problem class that generalizes many combinatorial problems including bin-packing

---

\*Universities Space Research Association

†The authors gratefully acknowledge Sailesh Ramakrishnan, Ronen Brafman and Michael Freed for reviewing our early drafts and providing editorial assistance. This work was funded by the NASA Intelligent Systems Program.

(Chapters R-26, R-27 and R-28), prize collecting TSP (Chapter R-29) and scheduling (Chapters R-33, R-36 and R-37). There are numerous extensions to classical planning that capture models of uncertainty in the world and in action outcome; readers interested in learning more about planning are referred to [TGN04].

## 1.1 Classical Planning

The core of classical planning problems is goal achievement; given a particular state of the world, and actions that can manipulate the state, the problem is to find a sequence of actions that lead to one of a designated set of goal states. More formally, a *planning domain*  $D$  consists of a set of world states  $W$  and a set of actions  $A$ . An action  $a$  defines a deterministic mapping  $a : W \rightarrow W$ . A *planning problem instance*  $\langle D, i, G \rangle$  consists of a planning domain  $D = \langle A, W \rangle$ , an initial state  $i \in W$  and a set of goal states  $G \subset W$ . The set of actions define a directed graph on  $W$ ; thus, the problem is to find a path from  $i$  to a state  $g \in G$  or prove that no such path exists. Optimal planning has costs associated with the use of actions, or reward associated with goals; the objective is to find a path that minimizes an overall cost function.

When described this way, planning appears quite easy; the problem is either to decide whether there is a path from  $i$  to any node  $g$ , or to find a minimal cost path. However,  $W$  and the associated set of actions  $A$  are usually described *implicitly* by exponentially compressing the description of  $W$  and  $A$ . The baseline problem description in classical planning is called *STRIPS*. The STRIPS formalism [FN71] uses a set of propositions  $P$  to implicitly define the set of world states;  $W = 2^P$ , with each state  $w = \{p_1, \dots, p_n\}$  being interpreted as the conjunction  $(p_1 \wedge \dots \wedge p_n)$ . Each action  $a$  is described by a set of preconditions  $pre(a)$  (propositions that must be true to enable  $a$ ), a set of added effects  $add(a)$  (propositions that  $a$  makes true) and a set of delete effects  $del(a)$  (propositions that  $a$  makes false). An action  $a$  is applicable in a state if the propositions in  $pre(a)$  are true in that state. The value of a proposition that does not appear in  $add(a)$  or  $del(a)$  is unchanged by the application of  $a$ .

Given this description of a planning domain, a STRIPS planning problem is defined by a single initial state  $i \subseteq P$ , and a set of goal propositions  $g \subseteq P$ , implicitly defining a set of goal states  $G = \{v \in 2^P | g \subseteq v\}$ . A plan can be viewed as an ordered sequence of actions. A plan can also be viewed as an ordered sequence of concurrent action *sets*; two actions  $a, b$  may be *concurrent* in a plan if  $(pre(a) \cup add(a)) \cap del(b) = \emptyset$

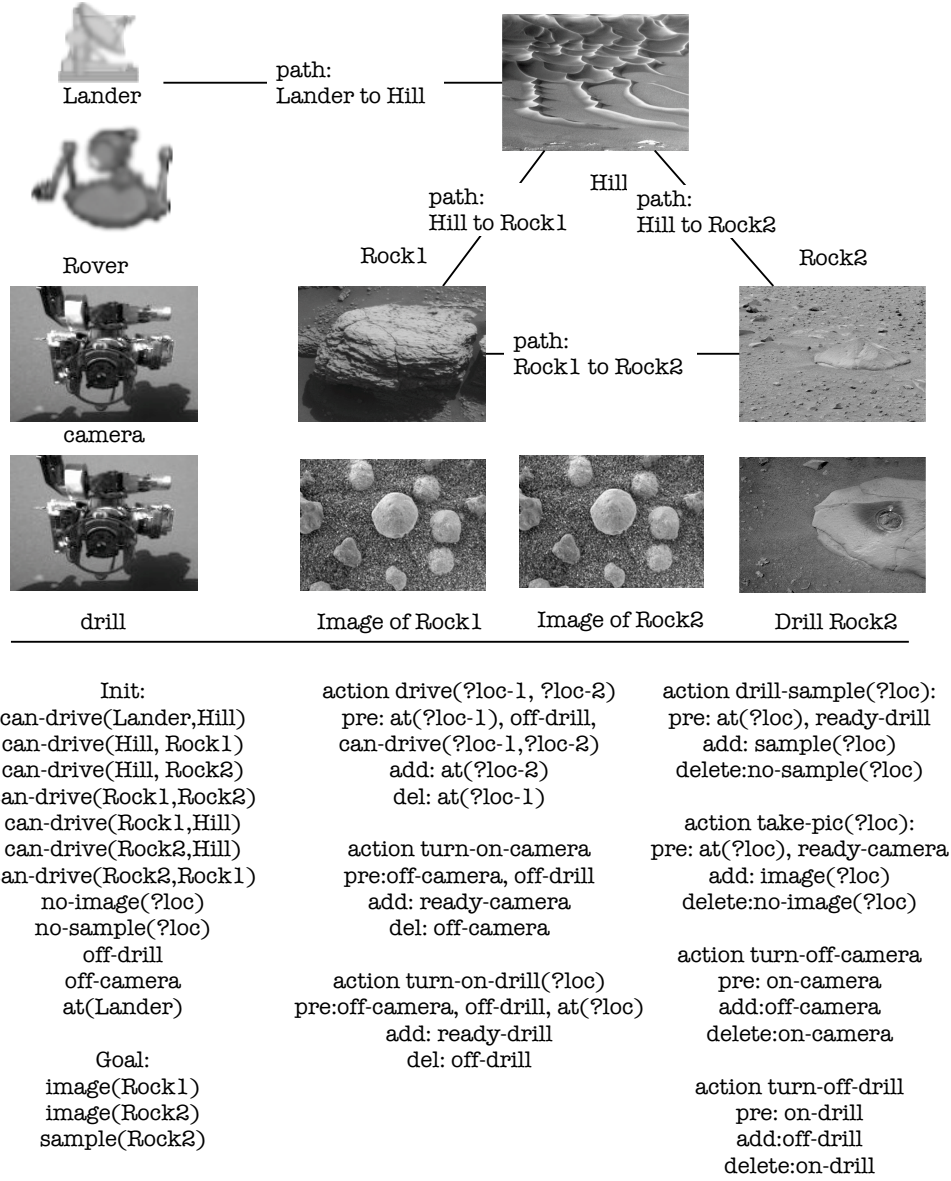


Figure 1.1: The STRIPS encoding of a simple planetary Rover domain. The **can-drive** propositions describe the simple road network in the picture. The rover is permitted to drive with the camera on, but cannot drive with the drill on. The drill and camera cannot both be turned on simultaneously. The rover must be **at** a location in order to take a picture of sample.

and  $(pre(b) \cup add(b)) \cap del(a) = \emptyset$ . A plan is valid if, for each action set, all actions in the set may be concurrent, and if every action in the set is applicable in the state resulting from the sequential application of the previous action sets. The core decision problem is to find a valid plan, i.e., a sequence of applicable actions sets,  $(A_1, \dots, A_n)$ , that maps the initial state to a goal state.

Figure 1.1 shows a simple STRIPS version of a planetary rover domain. The domain consists of a mobile rover with two instruments, a camera and a drill. In this example, the propositions are described using a combination of predicates and parameters; for example, the predicate `at` takes a location parameter. Initially, the rover is at the `Lander`, it has collected no images or samples, and its drill and camera are off. The rover can drive from the `Lander` to a `Hill`, and then to one of two rocks. The rover can take pictures or sample with the drill, but not both at the same time. The rover cannot drive with the drill on. Finally, the rover's goals are to take a picture at `Rock1`, and both a picture and a drill sample at `Rock2`.

Optimal planning couples the constraints on reaching goals with a function mapping valid plans to values. Common cost functions are the makespan of the plan (i.e. the number of action sets in the plan), the number of actions in the plan, and the utility of a plan. The utility of the plan is the sum of the rewards for propositions in the goal state reached, minus the sum of action costs. Consider again Figure 1.1. Suppose we had two rovers at the lander instead of one; the goal is to gather an image of each rock, and to sample `Rock2`<sup>1</sup>. A minimum makespan plan would have two rovers go to `Rock2`, since one can drill and the other can take the image. Since the camera can stay on while driving, whichever rover took the image at `Rock2` could drive to `Rock1` and take the image there. By contrast, a minimum action plan would use only one rover, and would first go to `Rock1`, leave the camera on while going from `Rock1` to `Rock2`, then do the drilling operation. The minimum makespan plan has makespan 5 and uses 10 actions. The minimum action plan, by contrast, has makespan 9 but uses only 9 actions. It is clear how adding action cost and goal reward to these examples creates complex and interesting problems.

It should be noted that planning research has been moving towards more complex, representations of classical planning problems that compactly express constraints involving time, resources, and more complex planning domain rules. We will briefly discuss these extensions at the end of the chapter.

---

<sup>1</sup>The STRIPS domain would need to be extended to include action descriptions for the second rover; we omit this for brevity.

## 1.2 Methods for Solving Classical Planning Problems

Planning has motivated considerable work on approximate algorithms, local search techniques, exploitation of structure (e.g. divide-and-conquer methods), and the use of approximations as components of exact planning algorithms. The motivation behind this is the computational complexity of planning. The complexity is typically measured in the size of the domain, which is usually dominated by the size of the action descriptions  $A$ . Standard STRIPS problems, where  $A$  is finite, are  $\mathcal{PSPACE}$ -complete. As we will see in this chapter, restrictions on the problem specification can reduce the computational complexity. For example, the class of planning problems where the length of the plan is bounded is  $\mathcal{NP}$ -complete. In this section we give a broad overview of commonly used search methods to solve planning problems, which will set the stage for a detailed discussion of approximation and local search techniques.

Search strategies for planning can be characterized by the search space representation and algorithm for traversing the search space. One of the most commonly used strategies for planning is *state-space search*, in which the planner searches for a path between the initial state and a goal state. The state space and the set of possible transitions is both implicit and exponential in size. To save time and space, states and allowed state transitions are identified “on the fly” during planning. This leads to two primary state-space search strategies; *progression*, which involves searching from the initial state towards a goal state, and *regression search*, where the search starts with the propositions in the goal states and search backwards for a path to a state that contains only propositions that are true in the initial state.

Another strategy, *plan-space search*, searches the set of possible plans. Typically, a plan-space search algorithm starts with the empty plan as its current candidate plan. At each point in the search, if the candidate plan is not yet a valid solution, the algorithm identifies a flaw that prevents the candidate plan from being a solution, and searches over flaw resolution strategies. For example, a candidate plan may not be valid because a precondition for an action is not guaranteed to be true; one class of resolution strategies is the set of actions that adds the required precondition. Plan-space search is not restricted to progression or regression, but can generate arbitrary plans.

Solving planning problems with a systematic search algorithm requires heuristics to guide the search process. In the most general sense, *heuristics* provide guidance on how to make choices in the search process.

For planning problems, heuristics are primarily used to rank candidate next steps, with the objective of assigning higher values to steps that point towards the nearest valid plan, or the overall optimal plan. *Admissible* heuristics always under-estimate the distance to the goal in state-space. Admissible heuristics are appealing because, when used with breadth-first search, the first feasible solution found is also an optimal solution. In state-space search, this yields optimal plans. Unfortunately, there is a tradeoff between admissibility and accuracy, which is manifested in the speed of search.

In state space search, heuristics are evaluated on states or sets of propositions, and estimate the minimal distance to the search objective. For progression, this distance is from the current state to a goal state, while for regression search the distance is from the current set of propositions to a subset of the initial state. In plan-space search, heuristics are more complex since a plan need not define a single current state that can be compared to the goal. For example, a plan containing actions with missing preconditions implicitly identifies a *set of paths* through the state space, but none of these paths may be valid.

In section 2, we discuss a variety of uses of *relaxations* in planning algorithms. One common use of relaxations is to generate efficient, accurate heuristics to guide search; these techniques are discussed in Section 2.1. The solution of the simpler problem can be used to guide the search; for example, it can be used as input to a heuristic choice ranking function, or as an indicator of which choices to pursue and which choices to ignore. We will spend most of our time discussing relaxation based heuristics for state-space search, but will mention some methods used to guide plan-space search. Another use of relaxations is to *approximate* the solutions to planning problems. These methods are described in Section 2.2. Approximations are most often used for optimal planning problems, but sometimes are used for decision problems as well.

A second class of search guidance techniques relies on identifying and exploiting *problem structure*. These techniques are described in Section 2.3. Planning problems often contain *sub-problems* that can be solved by specialized combinatorial algorithms, e.g., packing, path planning, and scheduling. Another common form of structure exploitation is to *partition* the problem by either ordering or separating goals, thus creating smaller, more easily solved planning problems.

Local search is a powerful technique that has been used successfully on a variety of combinatorial problems. We describe local search approaches to planning in Section 3. Planning algorithms employing local

search can be applied directly to the space of plans; neighborhood selection for these algorithms can be constructed by using variants of the cost functions discussed in Section 2. Local search can also be done by transforming the space of plans into another space that is more amenable to existing local search algorithms.

## 2 Relaxations of Classical Planning Problems

Approximations are the foundation for many of the search guidance techniques used in planning. The basic idea is to automatically generate an approximation of the given problem, solve or analyze the approximation, and derive useful heuristic information from the result. Relaxations and related approximation techniques are particularly appealing, as they can be relatively easy to solve, while still retaining important aspects of the original problem, and thus being more likely to provide useful information. In this section, we examine relaxation and approximation techniques used to guide planners. We start with approximation-based heuristics, and then move on to the use of similar techniques for search space pruning. Finally, we examine methods that identify sub-problems in planning and use solutions to those to guide the search.

### 2.1 Heuristic Guidance Using Relaxed Planning Problems

Effective search guidance is essential to the success of systematic search methods for planning. For planning algorithms this means automatically calculating heuristics from the problem specification. The challenge is to make such heuristics accurate enough to guide search effectively while preserving speed.

We will begin by considering heuristic estimation in state-space planning. Since planning goals are stated only in terms of what propositions are true in goal states, the principal source of difficulty in finding plans is the effect of action delete lists. A *relaxed planning problem* is one where all the delete lists have been eliminated. While the existence of a plan can be determined in polynomial time for relaxed planning problems, finding a plan with a minimal number of actions, i.e., the path to the nearest goal, is still  $\mathcal{NP}$ -complete (the minimal action set problem is easily reduced to minimal set covering). Arbitrary solutions to the relaxed planning problem offer poor heuristic guidance, so more refined approximations are needed. One approach, used in progression, is to estimate the distance to individual propositions of a goal state separately, and then combine the estimates. This approach, introduced in the HSP and HSP2 planners [BG01], can be

described by a cost function  $c(s, p)$  that estimates the number of actions needed to make a proposition  $p$  true when starting in state  $s$ . The cost estimate, which is easily calculated in low-order polynomial time, is defined by:

$$c(s, p) = \begin{cases} 0 & \text{if } p \in s \\ 1 + \min_{a \in A, p \in \text{add}(a)} f(\{c(s', p) \mid s' \in \text{pre}(a)\}) & \end{cases}$$

Letting  $f = \max$  when combining costs yields a heuristic  $h_{\max}$  that is admissible, whereas letting  $f = \sum$  yields a heuristic  $h_{\text{add}}$  that may underestimate or overestimate the real cost. A variation on this idea is found in the FF planner [HN01], but instead of combining estimates for individual propositions, the full relaxed planning problem is solved in a way that biases it towards short solutions. While this does not optimally solve the relaxed problem, the results are often sufficiently accurate to provide a useful heuristic estimate.

Consider our sample domain with the initial state  $\text{at}(\text{Rock1}) \wedge \text{camera} - \text{off}$  and the goal of having  $\text{image}(\text{Rock2}) \wedge \text{at}(\text{Rock1})$ . The shortest plan has four actions, while both  $h_{\max}$  and  $h_{\text{add}}$  give an estimate of 3. It is worth noting that while  $h_{\max}$  is admissible,  $h_{\text{add}}$  is neither an upper or a lower bound. To see that, consider the same initial state, with the goal of having  $\text{image}(\text{Rock1}) \wedge \text{image}(\text{Rock2})$ . The shortest solution is a four step plan, while  $h_{\max}$  estimates 3 steps and  $h_{\text{add}}$  estimates 5 steps.

The same idea can be applied in regression search, but there are some notable differences in how things work. For one, since the distance estimate is always from the initial state to a set of propositions, the heuristic can largely be pre-calculated for each individual proposition, leaving only a simple combination operation; this idea is discussed in [BG01]. However, this simple heuristic will often permit much of the regression search effort to be spent on finding paths from sets of propositions which cannot be reached from the initial state, as they contain two or more propositions that are mutually exclusive. This has led to efforts to identify mutually exclusive sets of propositions and thus prune such states from the search.

The notion of plan-graph [BF95] has turned out to be one effective approach to calculating mutual exclusions. A plan-graph is an approximation of the state space that can be calculated easily and represented in a compact fashion. The plan-graph is constructed as follows. The first level consists of the propositions defining the initial state. The next level consists of all *applicable* actions (an action  $a$  is applicable if all propositions in  $\text{pre}(a)$  appear in the previous level, and no two are marked as mutually exclusive). Actions that cannot be concurrent (as described in Section 1.1) are marked as being mutually exclusive. The following

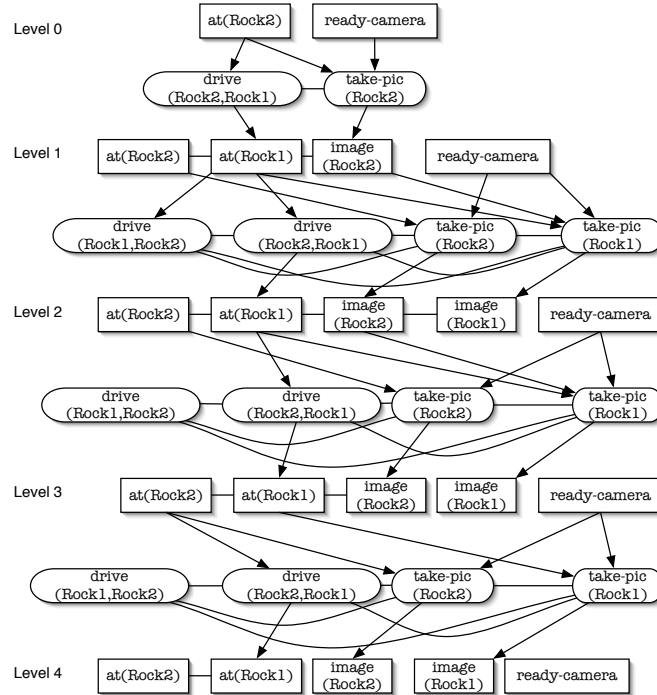


Figure 1.2: Part of the PlanGraph for the Rover domain. Boxes represent propositions, ovals represent actions. Arrows represent action preconditions and effects, while arcs represent mutual exclusions. We show the PlanGraph assuming the rover is `at(Rock1)` and the camera is ready. We only show actions `drive` and `take-pic` for simplicity. Notice that the set of mutual exclusions does not change after level 4 of this PlanGraph.

level consists of all the propositions in the initial level, as well as all effects from actions in the second level. Propositions pairs that can only be achieved by mutually exclusive actions are marked as such. This process continues until the set of fluents and mutual exclusion annotations becomes stable. An example of a partial plan graph for the simple Rover domain is shown in Figure 1.2.

The approximation provided by plan-graph is a powerful tool for guiding search methods. In the Graph-plan planner [BF95], a simple plan-space regression search is used, with no particular emphasis on action selection heuristics. Even so, the mutual exclusion annotations significantly reduce the effort spent on inconsistent states. The plan-graph has also been used successfully in state space search, as they provide a useful

approximation of the search space and offer a way to identify impossible states in regression search. Furthermore, plan-graphs have been used to formulate planning problems as constraint satisfaction problems, [LB03] [DK00], and as satisfiability problems [KMS96] (discussed in Section 3).

In many planning algorithms, and especially in state space search, the plan graph approximation has been used to generate heuristics. A simple admissible heuristic calculates the distance to a state based on the level where the propositions in the state first appear without mutual exclusion annotations. For example, in Figure 1.2, we can see that the first time `image(Rock1)` and `at(Rock2)` appear together without being mutually exclusive is at level two. The state space heuristics describe above, and the plan-graph-based heuristics are in fact related special cases of a more general heuristic [HG00]. This generalized heuristic is based on calculating the number of actions required to achieve a set of propositions of bounded size  $b$ . If  $b = 1$ , then the heuristic is the HSP heuristics; if the  $b = 2$ , it gives the plan graph heuristic.

The power of the plan-graph has led to many plan-graph-based heuristics. Simple heuristics often underestimate the number of actions needed to reach a goal, so efforts are typically aimed at improving that. One approach is to partition a set of propositions into minimally interacting subsets and then add up the estimates calculated for each subset. The plan-graph can be used to estimate the interactions between actions and take those into account in heuristic estimates [NKN02].

We note that plan-graph -based heuristics have also been successfully applied to controlling search for certain types of plan-space planners [NK01]. The idea is to use a special progression planner that focuses search on a limited set of applicable actions. This limitation simplifies the mapping of partial plans to states, making distance estimates easier to calculate.

To this point, we have examined abstractions based on relaxing action effects, and using the resulting problems to derive heuristics. Another approach to relaxing planning problems is to eliminate propositions from the problem description. This idea is used in planning with pattern databases [Ede02]. An entry in the database maps a state to a heuristic evaluation that is calculated as the cost of solving a planning problem that is restricted to a subset of the propositions in the planning problem. Multiple such entries, each using a planning problem restricted to a different set of propositions, can be combined into an overall admissible heuristic estimate. The selection of proposition sets is crucial to the value of each entry; typically, the subsets

are chosen such that each member has minimal interactions with propositions outside the subset.

Optimal planning problems lead to another set of variations on this theme. These problems require action costs and goal rewards to be folded into the standard plan graph or state space search distance estimates. In  $\text{AltAlt}^{PS}$  and  $\text{Sapa}^{PS}$  [vdBNDK04], which are progression search and regression planners respectively designed to solve optimal planning problems, the action cost  $C(a, t)$  and proposition costs  $C(p, t)$  at each level  $t$  of the plan-graph are calculated by:

$$C(p, t) = \min(C(a, t - 1) + \text{cost}(a))$$

$$C(a, t) = f(\{C(s', t) : s' \in \text{pre}(a)\})$$

where  $f$  is either  $\min$  or  $\sum$ , and the cost values are initialized with  $C(p, 0) = 0$  if  $p \in i$ ,  $C(p, 0) = \infty$  if  $p \notin i$ , and  $C(a, 0) = \infty$ .

Finally, Linear Programming approximations that relax variable domains from the propositional set  $\{0, 1\}$  to the interval  $[0, 1]$  can be used on planning problems. These approximations can be used to guide the search and to prune irrelevant or suboptimal parts of the search space, as is done in many standard integer programming techniques; examples of this technique include [Byl97] and [VBLN99].

## 2.2 Solution Approximations Using Relaxation

While heuristics have been found to help a great deal when solving planning problems, there are many cases when they are not sufficient for solving a problem effectively. This can stem from the heuristics not providing enough information to steer the search effort, or the fact that a significant portion of the search space must be searched to solve a given problem, such as is the case when finding optimal plans or proving that no solution exists. To address these problems, many have turned to methods that reduce the search by *pruning* parts of the search space, often using approximate methods.

Approximate pruning techniques have been explored for most existing planning formulations, but they vary greatly depending on problem being solved and formulation being used. One common notion is to map the problem to an easier class of problem. In those cases, the approximation primarily involves bounding some aspect of the planning problem, such as the length or number of actions. Another common approach is to discard candidates that are viewed as being unlikely to lead to a solution. For example, in the FF planner

[HN01], one method is to limit action selection to “helpful” actions that achieve a desired goal or sub-goal. A variation of this approach is to limit the number of options considered at each point.

Optimal planning is a fertile area for approximations for search space pruning, as guaranteed optimality is a particularly difficult problem, necessitating the use of methods that balance the search effort with the quality of the solution returned. The simplest approximation is for optimal planning problems where the solution value is a function of which goal propositions are achieved. Selecting a subset of goal propositions up front turns the problem into a standard planning problem. For example, in  $\text{AltAlt}^{PS}$  [vdBNDK04], the set of goals is generated iteratively. For each goal  $g$  not yet in a current goal set  $G'$ , solve the relaxed planning problem for goals  $G' \cup g$ , biased towards reusing the relaxed plan for achieving  $G'$ . Then select the one that adds the most value. A more sophisticated variation of the selection strategy examines subsets of candidate goals that are mutually consistent [NK05].

Another approximation technique is to solve an easier planning problem first, then modify the solution to meet the more difficult objective. This is particularly prevalent in problems where plan length (makespan) must be minimized. One approach to minimizing makespan is to “parallelize” a plan minimizing the number of actions [DK03]. Two approximations are at work in this technique; not only is finding a minimal length plan from a given sequential plan  $\mathcal{NP}$ -complete [Bäc98], but also an optimal solution to the sequential planning problem does not necessarily provide the best set of actions for a minimal length parallel plan. This is demonstrated by the two rover example in Section 1. Other approaches have included adding constraints to limit the search space [BR05] or heuristics [HN01] to influence a search technique that otherwise ignores the desired plan optimization criterion.

### 2.3 Heuristic Guidance Via Sub-Problem Identification

Structure and sub-problem identification is another powerful tool for guiding search. One class of techniques involves identifying common combinatorial problems as sub-components of planning problems. Such problems can then be solved with specialized, efficient solvers. The other class of techniques identifies structures like partitions and ordering restrictions, which are then used to split the planning problem into smaller, nearly independent planning problems that can be solved quickly with general-purpose planners, after which

the resulting plans are easily merged.

The best known technique for combinatorial sub-problem identification uses *type extraction* [LF00] to identify sets of related propositions and the relations between them. This is done by building up state-machines whose nodes are propositions, and whose transitions are actions. For example, the characteristics of a transportation domain are a set of location propositions,  $\text{at}(\mathbf{x})$ , and a mobility action  $\text{move}(\mathbf{x}, \mathbf{y})$  that enables a change in location. These techniques can identify interesting and non-obvious sub-problems; an example in [LF00] shows how a wall-painting domain leads to a transportation sub-problem due to constraints on what colors walls can be painted; i.e. the walls are mobile along a map of colors. Type extraction is also extensible to “dependent” maps (for example, travelers can only travel places with passports), transported objects (packages carried in trucks), and multiple move propositions (drive and fly operators). Furthermore, the approach can be extended to other sub-problem types such as packing or scheduling. A significant disadvantage of these techniques is that humans must write the sub-problem descriptions. And the issue of how to handle multiple sub-problems in the same problem instance is an open problem.

The notion of identifying state machines within planning problems is a powerful idea. A more general approach to identifying problem structures relies on translating planning problems into SAS+ [BN95], using a transformation algorithm described in [EH99]. In this representation, a planning problem consists of finite domain variables, and actions cause subsets of these variables to change values; each variable can be thought of as a state machine. A SAS+ representation of the Rover domain is shown in Figure 1.3. A SAS+ domain’s *causal graph* is a directed graph over the variables of the problem; there is an edge from  $u$  to  $v$  if a precondition on  $u$  causes  $v$  to change value, or an effect causes both  $u$  and  $v$  to change value. [Hel04] Helmert identifies a subclass of  $\mathcal{NP}$ -complete SAS+ problem instances whose causal graph has one variable with no out-edges, called a high-level variable, and for which goals are defined only for a subset of high-level variables. A polynomial time approximate algorithm for these problems enumerates paths of value transitions from shortest to longest; each path is checked to ensure it satisfies the goal, and that the action descriptions for the low-level goals are satisfied. These plans are converted into distance estimates by summing the number of value transitions for the high-level variable plan and the shortest paths between the initial and final values for the low-level variable.

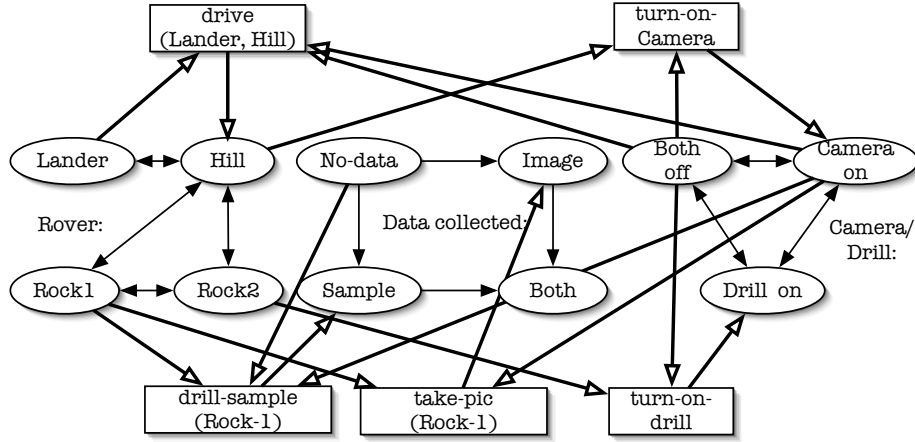


Figure 1.3: The Rover domain in the SAS+ formalism of Bäckström. Only one instance of each action is provided for simplicity. Ovals are values of state variables, boxes represent actions. State variable value transitions are shown with black arrows, action preconditions and effects are shown with white arrows.

When it comes to planning problems involving transportation and optimization, a common sub-problem is the Orienteering problem or Prize-collecting TSP (for more information on this problem see Chapter R-29). The Orienteering problem is quite simple compared to the an optimal planning planning problem, consisting of a graph with weighted nodes and edges, where the node weights correspond to edge traversal costs, while node weights correspond to rewards for visiting each edge. Methods for constructing and using the Orienteering problem for optimal planning are described in [Smi04]. The extraction of the Orienteering problem consists of three phases. The first phase involves a sensitivity analysis step, in which the ground propositions are analyzed for the variable cost of goal achievement. This is done by constructing a relaxed plan for each goal  $g$  and calculating costs using the same formula for calculating costs for AltAlt<sup>PS</sup>. For each proposition  $p$  in the final plan for  $g$ , assume  $p$  was true initially, and that propositions mutually exclusive with  $p$  are infinitely expensive, then recalculate the cost. The “seed” of the Orienteering problem comprises the set of all propositions for which the cost of goal achievement varies significantly (using a parameter passed

to the algorithm). The problem is complemented by applying each action to each possible state which is both reachable from the initial state, and consists only of propositions in the seed set; this is calculable using a normal plan-graph. The plan-graph is used to estimate the costs of achieving goals at each state of the resulting problem using the same cost function; this process adds new nodes (states derived from the plan-graph) and edges. Approximate solutions to the resulting Orienteering problem then provide heuristics to a classical planner, which can either be an approximate or complete planner.

Another class of structure identification and exploitation techniques relies on identifying sub-problems that are solved with a planner instead of special-purpose combinatorial solvers. This approach relies on identifying small sub-problems and guaranteeing that an overall solution can be constructed from the sub-problems. One method involves finding *landmarks* [HPS04], which are propositions that must hold in all valid plans. These propositions can be goals or intermediate states that must be reached to satisfy the goals. Two landmarks are *necessarily ordered* if one is a precondition for achieving the other. Again referring to Figure 1.1, the goals of imaging **Rock1** and **Rock2** from the initial state of rover at **Lander** both require being at the **Hill** at some point. Deciding if a proposition is a landmark, and deciding whether landmarks are necessarily ordered, are *PSPACE*-complete problems. A subset of landmarks are found by building the relaxed plan-graph; initially the goals  $g$  are landmarks, and subsequently every proposition needed by all actions is a landmark. The relaxed plan-graph is then used to find a subset of the necessary orders. The resulting identified landmarks and orders are used to provide search control to planners, either by controlling decision making or providing guidance to sequential divide-and-conquer algorithms.

### 3 Planning Using Local Search

Local search is a term used to describe a broad range of strategies for solving combinatorial problems. Local search is appealing because it tends to solve such problems fast, but theoretical analysis of average case performance of such algorithms is difficult, and considerable work is often required to tune these algorithms for best performance. In this section we describe a variety of local search techniques for planning.

### 3.1 Local Search Algorithms

Local search algorithms are characterized by a candidate solution to a problem, a set of operators that modify that solution in order to produce a *neighborhood* of solutions, and a procedure that chooses one of the neighbors as the next candidate solution. When local search is used to solve decision problems, minimizing the number of violated constraints transforms the problem into a pure optimization problem. The most common strategy to solve such problems is a “greedy” one, in which the lowest cost neighbor is selected as the next candidate solution. Greedy local search algorithms can be trapped in *local minima* in which there are no improving moves; numerous strategies for detecting or avoiding local minima have been studied, from injecting random moves and random restarts into search (see Chapter R-17) to adding either temporary constraints (see Chapters R-16,R-23 on tabu search) or permanent ones that influence subsequent steps. Another problem is that of *large neighborhoods*, which can be expensive to evaluate. This leads to *early termination* rules that improve the rate at which moves are made, at the possible cost of slow improvement in the optimization criteria (see Chapter R-18 on Very Large Neighborhood search).

Constrained optimization problems can also be transformed into pure optimization problems by assigning a penalty to violated constraints. In constrained optimization problems over continuous variables, these penalties are called Lagrange Multipliers (see Chapter R-2). The theory of Lagrange Multipliers states that saddle points in the space of the original variables plus the Lagrange Multipliers are solutions to the original constrained optimization problem, as long as some other properties are satisfied; this theory has been extended to discrete problems as well.

### 3.2 Local Search Neighborhoods for Planning

A common method of local search for planning is to define a set of operators that directly manipulate plans, in effect performing local search over plan space. These methods generate a set of possible plans from the current plan. The plans are then assessed to determine whether they are an improvement over the current plan; this assessment drives selection of the next plan in the neighborhood.

One option is to operate on the space of partially ordered plans. When feasible plans are easy to find and optimization is required, local search over the space of plans is a sensible approach. Planning By Rewriting

or PBR [AK01] is one such technique. The *plan rewrite rules* are domain-specific operators that are designed to create better plans. Rules come in four classes: action reordering rules, collapsing rules (that reduce the number of actions), expanders (inverse of collapsers), and parallelizers (that eliminate ordering constraints). A sample reordering rule from the Rover (Figure 1.1) is: if the plan moves the rover to accomplish a goal, then moves back to the current position to accomplish another goal, reorder these goals and discard the extra movements. If feasible plans are needed, the violated rules can be assigned penalties. This approach is used by [WC04]; specifically, an elaboration on the landmark technique described in Section 2 is used to divide the plan into segments. The rules of the domain ensure that the plan is well-formed, that is, each segment is well-formed, and the actions in segment  $i$  establish the preconditions for states in segment  $i + 1$ .

Another option is to operate on the space of plan-graphs. An Action Graph  $C$  is a subset of a PlanGraph such that if an action is in  $C$  then the propositions in the precondition and add effect lists are also in  $C$ . Action graphs can have inconsistencies, either in the form of mutual exclusions or propositions not supported by actions. This leads to a natural local search neighborhood; actions can be added to establish support, or removed to eliminate unsupported actions or resolve mutual exclusions. Linear Action Graphs are action graphs with at most one action per level. Local search on linear action graphs only manipulates actions to fix unsupported precondition inconsistencies; this leads to a simpler and smaller search neighborhood. Actions may be added at any level; the resulting linear action graph is grown by one level. Actions with unsupported preconditions may also be removed. LPG [GSS03] is a local search planner that operates on Action Graphs or Linear Action Graphs. In Figure 1.4 we show a linear action graph neighborhood for the Rover example. We note that there is only way to legally insert the **turn-on-drill** action. In general, however, there may be many neighbors to evaluate. Due to the expense of evaluating the neighborhood, and its potential size, small subsets of inconsistencies may be used to generate neighbors, with heuristics guiding inconsistency selection. Further limits can also be imposed to reduce the neighborhood size.

The second method involves *transforming* the plan space into another representation and performing local search in this representation. One possible representation is SAT; considerable work on local search (and complete search) has been done on this problem. Propositions encode assertions such as “plan proposition holds at step” or “action occurs at step”, and clauses encode the domain rules governing legal action

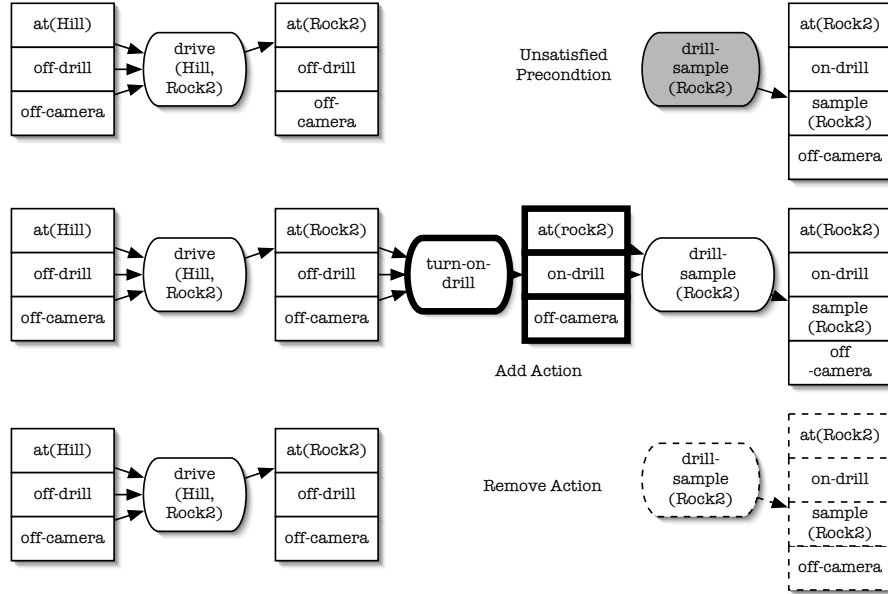


Figure 1.4: An example of the search neighborhood for LPG in the Rover domain using Linear Action Graphs. The preconditions for **drill-sample** are unsatisfied, inducing a search neighborhood of size 2. There is one legal place for adding **turn-on-drill**; alternately, **drill-sample** can be removed.

occurrences and their implications. The transformation of the plan space into SAT requires searching over a restricted set of plans; this is often done in a manner similar to construction of the plan-graph. Different encodings impact the size of the SAT problem and the local search topology, and therefore solver performance. One encoding, due to [KS92], works like this: actions imply their preconditions at the current instant and their effects at the next instant; one action occurs per instant; the initial state holds; and the frame conditions hold. An example of this SAT encoding for our Rover domain is shown in Figure 1.5. Another representation, described in [KS99], allows concurrent actions, and eliminates the action propositions from the encoding; all that remains are logical clauses expressing the possible transformations of propositions truth values after a (set of) action invocations. This reduces the size of the domain encoding and thus improves SAT solver performance. Other techniques to manipulate the SAT encoding include using unit propagation and posting single-and double-literal assertions to generate new clauses.

Actions imply preconditions and effects	Only one action at a time	Frame axioms (summarized)
drive-Hill-Rock1-i => at-Hill-i ^ can-drive-Hill-Rock1-i ^ at-Rock1-i+1 ¬at-Hill-i+1	drive-Hill-Rock1-i v ¬turn-on-Camera-i v ¬Sample-Rock1-Camera-i ... ^ ¬drive-Hill-Rock1-i v ¬turn-on-Camera-i v ¬Sample-Rock1-Camera-i ... ^ ¬drive-Hill-Rock1-i v turn-on-Camera-i v ¬Sample-Rock1-Camera-i ...	take-pic-Rock1-Camera-i ^ true-prop-i ^ => true-prop-i+1  drive-Hill-Rock1-i ^ true-prop-i => true-prop-i+1  turn-on-Camera-i ^ true-prop-i => true-prop-i+1 ...
turn-on-Camera-i => off-Camera-i ^ off-Drill-i ^ ready-Camera-i+1 ^ ¬off-Camera-i+1		
take-pic-Rock1-Camera-i => at-Rover-Rock1-i ^ ready-Camera-i ^ image-Rock1-i+1 ...		

Figure 1.5: The SATPlan encoding of the Rover problem.

SAT is a good representation for finding feasible plans, but more expressive representations can provide more flexibility to solvers. One such representation is a special subset of ILPs called OIPs (Overconstrained Integer Programs). The OIP consists of two systems of linear inequalities, one set  $A_i x \leq b$  defining feasible plans, and another set  $C_i x \leq d$  defining optimization criteria. The optimization function is to minimize the nonlinear function  $\sum_i C_i x - d | C_i x > d$ , that is, the “deviation” of the current solution from the optimal solution. The encoding is an extension of the SAT-based encodings described previously; it is easy to transform SAT into a 0 – 1 LP. Linear objective functions based on states and actions in the plan are easily encoded as systems of LPs. This representation is used by WalkSAT(OIP) [KW99].

### 3.3 Selecting the Successor

Local search algorithms that operate directly on the plan-state face the problem of deciding which neighbor to select. When the set of plans in the neighborhood is mostly feasible, as is the assumption behind PBR, the neighborhood can simply be evaluated by the function that determines the value of feasible plans. When the neighborhood consists of infeasible plans, as occurs with LPG, the problem is how to estimate the relative

merit of infeasible plans. One solution is to use the heuristics that estimate the amount of work needed to fix the problems with this plan. The situation is well summarized when considering heuristics estimating the work needed to find a feasible Linear Action Graph. Let  $threats(a, C)$  be the set of supporting preconditions of actions in  $C$  that become unsupported if  $a$  is added to  $C$ ;  $nopre(a, C)$  be the set of unsupported preconditions of  $a$  in  $C$ ; and  $unsup(b, C)$  be the set of supporting preconditions of actions in  $C$  that become unsupported if  $b$  is removed from  $C$ . A simple neighborhood evaluation function is

$$E_0^i(a, C) = |threats(a, C)| + |nopre(a, C)|$$

$$E_0^r(b, C) = |unsup(b, C)|$$

where  $E_0^i(a, C)$  if  $a$  is added to  $C$ , and  $E_0^r(b, C)$  is used if  $b$  is removed from  $C$ . A more complex heuristic recursively adds actions to support unsupported facts, using  $E_0^i(a, C)$  and  $E_0^r(a, C)$  to penalize these actions. The most complex heuristic extends the linear action graph using a regression planner that ignores delete effects of actions; the heuristic then counts actions in the resulting (relaxed) plan and conflicts between the relaxed plan and the current linear action graph.

Representations based on either SAT or constraints, such as OIP, induce simple neighborhoods. The common neighborhood in the SAT solvers used in SATPlan [KS96] is to reverse the assignment of each proposition. For constraint-based representations like OIP, the set of variable assignments satisfying one violated constraint form the search neighborhood. Representations that partition constraints into feasibility constraints and optimality constraints such as OIP enable algorithms that choose between sets of violated constraints according to a pre-determined bias.

Since the theory of Lagrange Multipliers has been extended to discrete search, it is possible to transform an optimal planning problem into a discrete constrained optimization problem, and implement a search to find extended saddle points in the Lagrangian space of a problem. This is the approach taken in [WC04]. The Lagrangian penalties are applied both to actions that apply as well as the constraints implied by the segmentation of the plan. The search for a saddle point utilizes numerous methods to control growth of the Lagrangian penalties, as well as the usual methods of neighborhood cost control and randomization to promote exploration. Additional methods are used to adjust segment boundaries during search. This approach can use many different planners to generate moves during search. This method is more general

than using OIPs, but still requires careful modeling of the search space.

## 4 Planning Ahead: The Future of Approximations and Planning

Recent work in AI planning has focused on extending the core goal achievement problem to better represent the complexity of real-world problems. This includes handling temporal and resource constraints found in scheduling problems [SFJ00], as well as more complex relations between actions and states. PDDL, the most popular planning domain description language, has been extended to provide more direct support for temporally extended state and action, numeric fluents (meant to capture resource states), and planning with knowledge of future state (exogenous events). In addition, other variants of STRIPS have been developed to include explicit resource declarations, resource-impacting actions and allow resource state based preconditions, for example [Koe98].

Work is under way to extend existing techniques to handle the additional complexity, and to develop new approaches to represent and reason about plans. This work includes approximation methods, structure identification techniques, and local search, which will continue to play a crucial role in effective automated planning. For search guidance, plan-graph methods have been extended to include temporal information [SW99] and numerical fluents [Hof03]. Structure identification and exploitation techniques include the use of special-purpose methods for scheduling sub-problems [Sri00]. Local search approaches are being utilized for resource-constrained planning [TCS<sup>+</sup>04]. However, a great deal of work still needs to be done to reach the full potential of approximation techniques and local search methods for planning in the future.

## References

- [AK01] J. L. Ambite and C. A. Knoblock. Planning by rewriting. *Journal of Artificial Intelligence Research*, 15:207–261, 2001.
- [Bäc98] C. Bäckström. Computational aspects of reordering plans. *Journal of Artificial Intelligence Research*, 9:99–137, 1998.

- [BF95] A. Blum and M. Furst. Fast planning through planning graph analysis. In *Proceedings of the 14<sup>th</sup> International Joint Conference on Artificial Intelligence*, pages 1636–1642, 1995.
- [BG01] B. Bonet and H. Geffner. Planning as heuristic search. *Artificial Intelligence*, 129(1-2):5–33, 2001.
- [BGHH05] M. Boddy, J. Gohde, T. Haigh, and S. Harp. Course of action generation for cyber security using classical planning. In *Proceedings of the 15<sup>th</sup> International Conference on Automated Planning and Scheduling*, pages 12 – 21, 2005.
- [BJMR05] J. Bresina, A. Jonsson, P. Morris, and K. Rajan. Activity planning for the Mars exploration rovers. In *Proceedings of the 15<sup>th</sup> International Conference on Automated Planning and Scheduling*, pages 40 – 49, 2005.
- [BN95] C. Bäckström and B. Nebel. Complexity results for SAS<sup>+</sup> planning. *Computational Intelligence*, 11(4):625–655, 1995.
- [BR05] M. Buttner and J. Rintanen. Improving parallel planning with constraints on the number of actions. In *Proceedings of the 15<sup>th</sup> International Conference on Automated Planning and Scheduling*, pages 292 – 299, 2005.
- [Byl97] T. Bylander. A linear programming heuristic for optimal planning. In *Proceedings of the 14<sup>th</sup> National Conference on Artificial Intelligence*, pages 694–699, 1997.
- [DK00] M. B. Do and S. Kambhampati. Solving planning-graph by compiling it into CSP. In *Proceedings of the 5<sup>th</sup> International Conference on Artificial Intelligence Planning Systems*, pages 82–91, 2000.
- [DK03] M. B. Do and S. Kambhampati. Improving temporal flexibility of position constrained metric temporal plans. In *Proceedings of the 13<sup>th</sup> International Conference on Automated Planning and Scheduling*, pages 42 – 51, 2003.

- [Ede02] S. Edelkamp. Symbolic pattern databases in heuristic search planning. In *Proceedings of the 6<sup>th</sup> International Conference on Artificial Intelligence Planning Systems*, pages 274 – 283, 2002.
- [EH99] S. Edelkamp and M. Helmert. Exhibiting knowledge in planning problems to minimize state encoding length. In *Proceedings of the 5<sup>th</sup> European Conference on Planning*, pages 135–147, 1999.
- [FN71] R. Fikes and N. Nilsson. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2(3/4):189–208, 1971.
- [GPNV03] K. Golden, W. Pang, R. Nemani, and P. Votava. Automating the processing of earth observation data. In *Proceedings of the 7<sup>th</sup> International Symposium on Artificial Intelligence, Robotics and Space*, 2003.
- [GSS03] A. Gerevini, L. Saetti, and I. Serina. Planning through stochastic local search and temporal action graphs. *Journal of Artificial Intelligence Research*, 20(239-290), 2003.
- [Hel04] M. Helmert. A planning heuristic based on causal graph analysis. In *Proceedings of the 14<sup>th</sup> International Conference on the Automated Planning and Scheduling*, pages 161 – 170, 2004.
- [HG00] P. Haslum and H. Geffner. Admissible heuristics for optimal planning. In *Proceedings of the 5<sup>th</sup> International Conference on Artificial Intelligence Planning Systems*, pages 140–149, 2000.
- [HN01] J. Hoffmann and B. Nebel. The FF planning system: Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research*, 14:253 – 302, 2001.
- [Hof03] Jörg Hoffmann. The Metric-FF planning system: Translating “ignoring delete lists” to numeric state variables. *Journal of Artificial Intelligence Research*, 20:291–341, 2003.
- [HPS04] J. Hoffmann, J. Porteous, and L. Sebastia. Ordered landmarks in planning. *Journal of Artificial Intelligence Research*, 22:215 – 278, 2004.

- [KMS96] H. Kautz, D. McAllester, and B. Selman. Encoding plans in propositional logic. In *Proceedings of the 5<sup>th</sup> International Conference on the Principle of Knowledge Representation and Reasoning*, pages 374–384, 1996.
- [Koe98] J. Koehler. Planning under resource constraints. In *Proceedings of the 13<sup>th</sup> European Conference on Artificial Intelligence*, pages 489 – 493, 1998.
- [KS92] H. Kautz and B. Selman. Planning as satisfiability. In *Proceedings of the 10<sup>th</sup> European Conference on Artificial Intelligence*, pages 359–379, 1992.
- [KS96] H. Kautz and B. Selman. Pushing the envelope: Planning, propositional logic and stochastic search. *Proceedings of the 13<sup>th</sup> National Conference on Artificial Intelligence*, pages 1194 – 1201, 1996.
- [KS99] H. Kautz and B. Selman. Unifying sat-based and graph-based planning. In *Proceedings of the 16<sup>th</sup> International Joint Conference on Artificial Intelligence*, pages 318 – 325, 1999.
- [KW99] H. Kautz and J. P. Walser. State-space planning by integer optimization. In *Proceedings of the 16<sup>th</sup> National Conference on Artificial Intelligence*, pages 526–533, 1999.
- [LB03] A. Lopez and F. Bacchus. Generalizing graphplan by formulating planning as a CSP. In *Proceedings of the 18<sup>th</sup> International Joint Conference on Artificial Intelligence*, pages 954 – 960, 2003.
- [LF00] D. Long and M. Fox. Automatic synthesis and use of generic types in planning. In *Proceedings of the 5<sup>th</sup> Artificial Intelligence Planning Systems*, pages 196–205, 2000.
- [MNPW98] N. Muscettola, P. Nayak, B. Pell, and B. Williams. Remote agent: To boldly go where no ai system has gone before. *Artificial Intelligence*, 103(1-2):5 – 48, 1998.
- [MP02] C. E. McCarthy and M. E. Pollack. A plan-based personalized cognitive orthotic. In *Proceedings of the 6<sup>th</sup> Conference on Artificial Intelligence Planning Systems*, pages 243–252, 2002.
- [NK01] X. Nguyen and S. Kambhampati. Reviving partial ordering planning. In *Proceedings of the 17<sup>th</sup> International Joint Conference on Artificial Intelligence*, pages 459 – 464, 2001.

- [NK05] R. Sanchez Nigenda and S. Kambhampati. Planning graph heuristics for selecting objectives in over-subscription planning problems. In *Proceedings of the 15<sup>th</sup> International Conference on Automated Planning and Scheduling*, pages 192 – 201, 2005.
- [NKN02] X. Nguyen, S. Kambhampati, and R. Sanchez Nigenda. Planning graph as the basis for deriving heuristics for plan synthesis by state space and CSP search. *Artificial Intelligence*, 135(1-2):73–123, 2002.
- [RDF05] W. Ruml, M. B. Do, and M. Fromhertz. On-line planning and scheduling for high speed manufacturing. In *Proceedings of the 15<sup>th</sup> International Conference on Automated Planning and Scheduling*, pages 30 – 39, 2005.
- [SFJ00] D. Smith, J. Frank, and A. Jónsson. Bridging the gap between planning and scheduling. *Knowledge Engineering Review*, 15(1), 2000.
- [Smi04] D. Smith. Choosing objectives in over-subscription planning. *Proceedings of the 14<sup>th</sup> International Conference on Automated Planning and Scheduling*, pages 393 – 401, 2004.
- [Sri00] B. Srivastava. Realplan: Decoupling causal and resource reasoning in planning. In *Proceedings of the 17<sup>th</sup> National Conference on Artificial Intelligence*, pages 812 – 817, 2000.
- [SW99] D. Smith and D. Weld. Temporal planning with mutual exclusion reasoning. In *Proceedings of the 16<sup>th</sup> International Joint Conference on Artificial Intelligence*, pages 326–337, 1999.
- [TCS<sup>+</sup>04] D. Tran, S. Chien, R. Sherwood, R. Castaño, B. Cichy, A. Davies, and G. Rabbideau. The autonomous sciencecraft experiment onboard the eo-1 spacecraft. In *Proceedings of the 19<sup>th</sup> National Conference on Artificial Intelligence*, pages 1040 – 1045, 2004.
- [TGN04] P. Traverso, M. Ghallab, and D. Nau. *Automated Planning: Theory and Practice*. Morgan Kaufman, 2004.
- [VBLN99] T. Vossen, M. Ball, A. Lotem, and D. Nau. On the use of integer programming models in AI planning. In *Proceedings of the 16<sup>th</sup> International Joint Conference on Artificial Intelligence*, pages 304–309, 1999.

- [vdBNDK04] M. van den Briel, R. Sanchez Nigenda, M. B. Do, and S. Kambhampati. Effective approaches for partial satisfaction (oversubscription) planning. In *Proceedings of the 19<sup>th</sup> National Conference on Artificial Intelligence*, pages 562 – 569, 2004.
- [WC04] B. Wah and Y. Chen. Subgoal partitioning and global search for solving temporal planning problems in mixed space. *International Journal on Artificial Intelligence Tools*, 13(4):767 – 790, 2004.